10/665,156

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 April 2003 (24.04.2003)

PCT

(10) International Publication Number
WO 03/034228 A1

(51) International Patent Classification[7]: G06F 12/00, 17/30

(21) International Application Number: PCT/US02/33090

(22) International Filing Date: 17 October 2002 (17.10.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/347,919	18 October 2001 (18.10.2001)	US
60/347,901	18 October 2001 (18.10.2001)	US
10/271,244	15 October 2002 (15.10.2002)	US
10/271,194	15 October 2002 (15.10.2002)	US
10/271,157	15 October 2002 (15.10.2002)	US
10/271,156	15 October 2002 (15.10.2002)	US
10/271,423	15 October 2002 (15.10.2002)	US
10/271,047	15 October 2002 (15.10.2002)	US
10/271,402	15 October 2002 (15.10.2002)	US
10/271,410	15 October 2002 (15.10.2002)	US

(71) Applicant: BEA SYSTEMS, INC. [US/US]; 2315 North First Street, San Jose, CA 95131 (US).

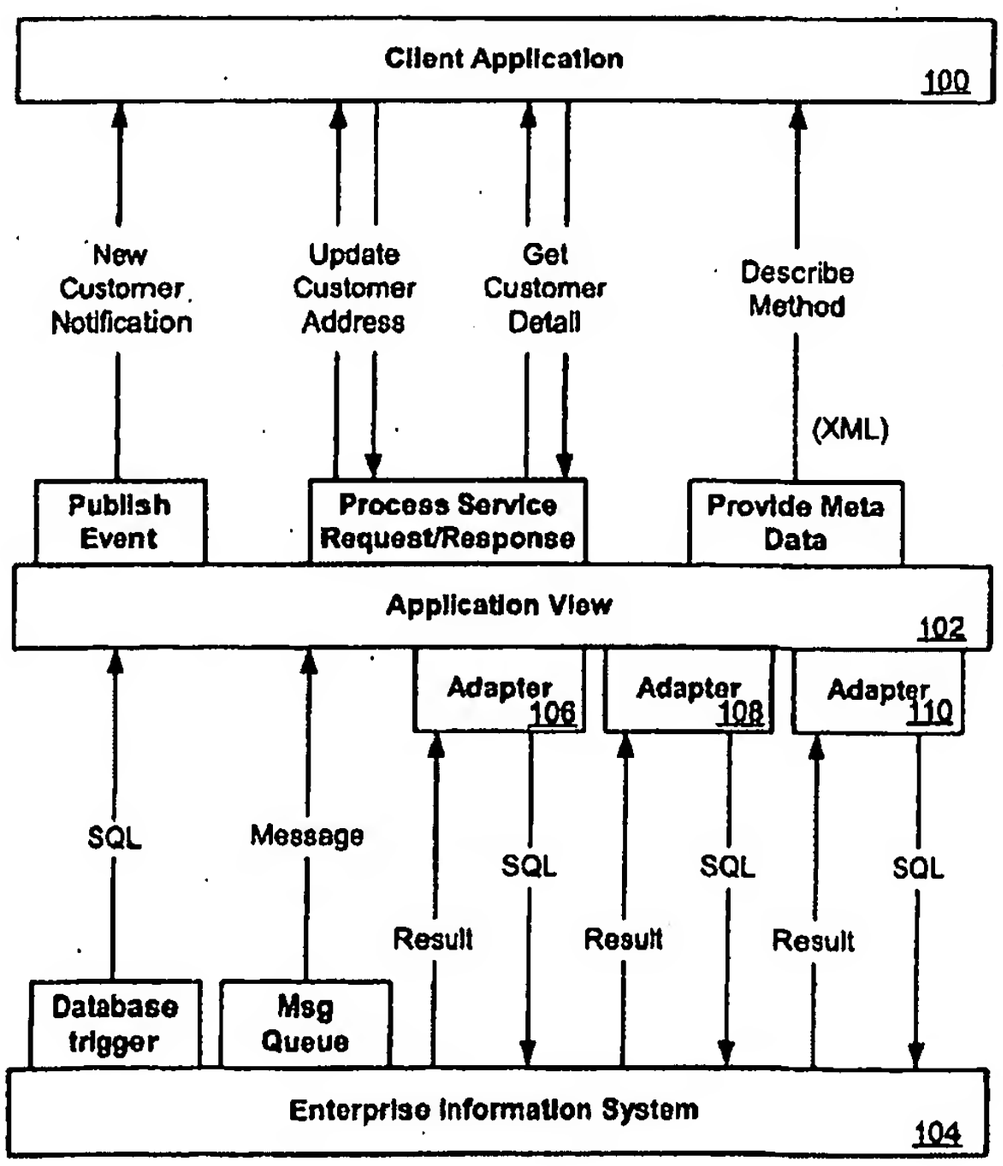
(72) Inventor: UPTON, Mitch; 10099 Briargrove Way, Highlands Ranch, CO 80126 (US).

(74) Agents: MEYER, Sheldon, R. et al.; Fliesler Dubb Meyer & Lovejoy LLP, Four Embarcadero Center, Suite 400, San Francisco, CA 94111-4156 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW,

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR IMPLEMENTING A SCHEMA OBJECT MODEL IN APPLICATION INTEGRATION



(57) Abstract: Communication can be passed between components, such as an enterprise system (104) and a client application (100), by utilizing schemas. A schema can ensure that a communication, such as a request or response, is in the proper format for one of the components. For instance, metadata can be received from an enterprise system (104) in response to a request from a client application (100). That metadata can be transformed into an XML document that conforms to an XML schema, such as by an XML schema mechanism. At least portions of the XML document can be validated against the XML schema, such as by using a schema object model. The XML document can be passed on to the client application (100) after validation.

WO 03/034228 A1

BEST AVAILABLE COPY

MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM AND METHOD FOR IMPLEMENTING A SCHEMA OBJECT MODEL IN APPLICATION INTEGRATION

CLAIM OF PRIORITY

This application claims priority to the following applications each of which is hereby incorporated herein by reference:

U.S. Provisional Patent Application No. 60/347,919, filed October 18, 2001, entitled "APPLICATION VIEW".

U.S. Provisional Patent Application No. 60/347,901, filed October18, 2001, entitled "EVENT ADAPTER".

U.S. Patent Application No. _____ entitled "APPLICATION VIEW COMPONENT FOR SYSTEM INTEGRATION," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD FOR INVOKING BUSINESS FUNCTIONALITY FOR A WORKFLOW," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD FOR IMPLEMENTING AN EVENT ADAPTER," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD USING A CONNECTOR ARCHITECTURE FOR APPLICATION INTEGRATION," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD FOR IMPLEMENTING A SCHEMA OBJECT MODEL IN APPLICATION INTEGRATION," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD UTILIZING AN INTERFACE COMPONENT TO QUERY A DOCUMENT," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD USING ASYNCHRONOUS MESSAGING FOR APPLICATION INTEGRATION," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD FOR IMPLEMENTING A SERVICE ADAPTER," by Mitch Upton, filed October 15, 2002.

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document of the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

CROSS-REFERENCED CASES

The following applications are cross-referenced and incorporated herein by reference:

U.S. Patent Application No. _____ entitled "APPLICATION VIEW COMPONENT FOR SYSTEM INTEGRATION," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD FOR PROVIDING A JAVA INTERFACE TO AN APPLICATION VIEW COMPONENT," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD FOR INVOKING BUSINESS FUNCTIONALITY FOR A WORKFLOW," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD FOR USING WEB SERVICES WITH AN ENTERPRISE SYSTEM," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD FOR IMPLEMENTING AN EVENT ADAPTER," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD USING A CONNECTOR ARCHITECTURE FOR APPLICATION INTEGRATION," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD UTILIZING AN INTERFACE COMPONENT TO QUERY A DOCUMENT," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD USING ASYNCHRONOUS MESSAGING FOR APPLICATION INTEGRATION," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEMS AND METHODS FOR INTEGRATION ADAPTER SECURITY," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. _____ entitled "SYSTEM AND METHOD FOR IMPLEMENTING A SERVICE ADAPTER," by Mitch Upton, filed October 15, 2002.

FIELD OF THE INVENTION

The invention relates generally to systems and methods for integrating applications.

BACKGROUND OF THE INVENTION

E-commerce has become a major driving factor in the new economy. To be successful in the long-term, e-commerce will require many companies to engage in cross-enterprise collaborations. To achieve cross-enterprise integration, a company must first integrate its internal applications. Using existing technology and tools, application integration can be an expensive proposition. No integration solution exists that is easy to use, affordable, and based on industry standards. Neither does a solution exist that is based on an industry standard infrastructure, has universal connectivity, is capable of massive scalability, and has accessible business process tools.

Application integration to this point has been very inward-focused. Many existing integration systems have not focused on integrating applications between enterprises. Even when integration solutions were used for cross-enterprise integration, the solutions were still narrowly focused and aimed at vertical markets. This inward focus did little to help companies field external business-to-consumer and business-to-business applications, such as applications that can utilize the Internet to generate revenue and reduce costs. The requirement for Internet-enabled applications led to the rise of the application server market. To date, application servers have primarily been used to host external applications targeted at customers and partners. Application servers are themselves packaged applications that, instead of solving a specific problem, are general-purpose platforms that host vertical solutions.

The first attempts at application integration were primarily focused on low-level implementation details such as the format of the data, the byte ordering between machines, and character encoding. The focus on low-level data formats was necessary because, for the first generation of application integration solutions, there were no widely adopted standards for data encoding that could be deployed across multiple vertical applications.

The traditional approach involved connecting individual systems to, in effect, hardwire the systems together. This approach can be complex, as connecting different systems can require an intimate, low-level knowledge of the proprietary technologies of multiple systems.

Present integration systems, which have moved away from "hardwiring" systems together, still suffer from a lack of standards. Each

integration vendor typically provides a proprietary solution for application integration, message transformation, message formats, message transport, and routing. Not one of these systems to date has achieved significant market share to enable its technologies to become the de-facto standard. This lack of standards has given packaged application vendors little incentive to integrate these systems with their. Further, each of these integration systems or servers has its own proprietary API, such that packaged application vendors cannot leverage development beyond a single integration server. This fragmentation of the integration market has provided little financial incentive for third parties.

SUMMARY OF THE INVENTION

Systems and methods in accordance with embodiments of the present invention allow communication to be passed between components, such as an enterprise system and a client application, by taking advantage of schemas. A schema can be used to ensure that a communication, such as a request or response, is in the proper format for one of the components. For instance, metadata can be received from an enterprise system in response to a request from a client application. That metadata can be transformed to a response document that conforms to a schema. The document can be validated against the schema and passed on to the client application.

Other features, aspects, and objects of the invention can be obtained from a review of the specification, the figures, and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a diagram of an integration system that can be used in accordance with one embodiment of the present invention.

Figure 2 is a diagram of an integration system that can be used in accordance with another embodiment of the present invention.

Figure 3 shows a method that can be used with the systems of Figures 1 and 2.

Figure 4 is a flowchart showing a process that can be used with the systems of Figures 1 and 2.

DETAILED DESCRIPTION OF THE INVENTION

Application integration components can be used to integrate a variety of applications and systems, such as Enterprise Information Systems (EISs). Information technology (IT) organizations typically utilize several highly-specialized applications. Without a common integration platform to facilitate application-level integration, these applications cannot be integrated without extensive, highly-specialized development efforts.

Application integration can utilize adapters to establish an enterprise-wide, united framework for integrating any current or future application. Adapters can simplify integration efforts by allowing each application to be integrated with an application server, instead of requiring that each application being integrated with every other application.

The development and widespread acceptance of standards such as the Java 2 Platform, Enterprise Edition (J2EE) from Sun Microsystems, Inc. of Santa Clara, CA, as well as the eXtensible Markup Language (XML), has laid the groundwork for a standardized approach to the development of these adapters. Perhaps the most significant of these standards for application integration is the J2EE Connector architecture. The J2EE

Connector architecture provides a standardized approach for the development of adapters for all types of applications, from legacy mainframe applications, such as CICS from IBM, to packaged applications such as PeopleSoft, Siebel, and SAP. The adoption of such standards enables businesses to develop adapters that work on any J2EE-compliant application server, for example.

Integration Architecture

Application integration can build on this standardized approach in an application integration framework by providing a standards-based architecture for hosting J2EE Connector architecture-based adapters. Developers can build J2EE Connector architecture-compliant adapters and deploy these adapters, in the integration framework, to connect enterprise applications to an application server.

These adapters can be used to define business-focused interfaces to an EIS, the interfaces referred to herein as "application views" of the respective adapters. An application view can provide a simple, self-describing, consistent interface to services and events in an application. Application views can make use of an adapter for an EIS, making it possible to expose existing information systems as business services. Unlike adapters, however, an application view does not require users to have intimate knowledge of the EIS or the client interface for that EIS, such that non-programmers or technical analysts can use application views. An application view can provide a business-oriented way for business analysts to access enterprise data without worrying about the programmatic details defined in an adapter. These same users may be otherwise unable to use an adapter directly, due to a lack of familiarity with the EIS.

An application integration component directed at enterprise application integration can have several primary aspects. If the functionality of an EIS such as a PeopleSoft system or an SAP system is to be invoked, an implementation of the J2EE Connector Architecture can be used. If something occurs inside an EIS system, such as a trigger going off, an event can be generated. This event may, in some embodiments, need to be communicated to an external application. An event architecture in an application integration component can handle this communication.

Application Views

An application view can provide significant value to an application integration component. An application view can abstract away much of the complexity in dealing with an application, such as a backend EIS system. Application views can also simplify the way in which adapters are accessed. Application views can provide a layer of abstraction, for example, between an adapter and the EIS functions exposed by that adapter. Instead of accessing an EIS by direct programming a user can simply: edit an adapter's application views, create new application views, or delete any obsolete application view(s). A layer of abstraction formed by application views can help non-programmers maintain the services and events exposed by an adapter. Each application view can be specific to a single adapter, and can define a set of business functions on that adapter's EIS. After an adapter is created, a Web-based interface for the adapter can be used to define application views.

If an application view is used as a primary user interface for an adapter, a number of features can be included that are not commonly found in existing enterprise application integration technologies. Application views can, for example, use XML as a common language among

applications. Service and event definitions can be used to expose application capabilities. XML schemas can be used to define the data for services and events. Bidirectional communication can also be supported in adapters.

An application view can be an integral part of an integration framework. An application view can provide a view of the application capabilities exposed by an adapter that a user can customize to meet specific needs. A user can tailor an application view, for example, for a specific business purpose. As a result, the application view can provide an effective alternative to the "one size fits all" approach that many applications provide for the design of a client interface. An application view can be defined for only the business or other capabilities that are applicable for a specific purpose. The capabilities can be customized such as by naming, describing, and defining the data requirements.

In one example, shown in **Figure 1**, adapters **106**, **108**, **110** can be developed that allow a client application **100** to communicate with an Enterprise Information System **104** through the use of an application view **102**. A developer can begin by coding an adapter that exposes the functionality in the enterprise application that accesses enterprise data. The functionality the adapter exposes could, for example, update records in a database using SQL statements, or could request information from an SAP system using its BAPI or IDOC interfaces. A business analyst, working with the developer, can then define an application view of the adapter using an application view interface.

Another example of a system is shown in **Figure 2**. In this figure, a client application **200** can communicate with an EIS **218** through an

integration server **202**. The integration server can be, for example, a web server or application server **202**, and can be included in a cluster or integration system, represented here by the dotted line. The integration server can include an application view component **204** and a resource adapter **206** for the EIS **218**. An XML schema API **208**, useful in generating an XML schema **214** for the client application **200** using the schema object model **212** can be included in the integration server **202**. An XML document API **210**, which can provide an interface to an XML document using an IDocument component **216** such as IDoc class libraries, for example, can also be included on the integration server **202**. The schema object model **212**, XML schema **214**, and IDocument component **216** do not need to be contained on the integration server **202**, but should be accessible to the integration server.

An application view is an object, which can be implemented in one embodiment as a stateless session JavaBean. There can be a Java interface to the application view for the client application. A Java application can be custom coded to use that object, such as by passing XML in and receiving XML back. In addition, a business process manager component can be included that allows process engineers to define workflows, and allows application views to be invoked as business services. In a workflow, a callout can be made to an EIS to get information such as a customer's credit record. The fact that the application view is a Java object or enterprise JavaBean can be hidden from the process and designer.

A web services interface can also be used with an application view. A protocol such as SOAP can be used to invoke a web service. Another protocol that may be used includes UDDI, a platform-independent, open

framework for describing services, discovering businesses, and integrating business services using the Internet. A WSDL protocol can also be used, which is an XML format for describing network services . A web services layer can be provided on top of the application view so that any application view can be invoked as a web service.

In application integration, new application views can be hot-deployed against an existing EIS through a web-based interface. An application view is hot-deployed when it is deployed with the system running, without restarting the destination server. A new customer management tool for SAP, for example, can also be defined through a web browser.

Integration Framework

Application integration can utilize an integration framework, which can provide a systematic, standards-based architecture for hosting application views. Features of such a framework can include application views for exposing application functions and design-time graphical user interfaces (GUIs), such as web-based interfaces that can be used for creating application views. The integration framework utilizes adapters, instead of "hardwiring" enterprise systems together. Once an adapter is deployed for an EIS, other components and applications can use that adapter to access data on the EIS.

A framework in accordance with one embodiment of the present invention relies on XML as the standard format for messages. XML includes XSLT, a standard for transforming XML documents into other XML documents. XSLT is designed for use as part of XSL, which is a stylesheet language for XML. In XSLT, an XML document is used to specify the

operations to perform on a class of XML documents in order to transform the documents' structure and content. An XSLT transformation can make use of any of the operations built into the Java programming language, or can make use of custom operations written either in Java or in native code. An integration framework allows a business process to invoke an XSLT engine in order to transform XML messages.

An integration framework can also rely on standards for transporting messages such as Java Message Service (JMS) and HTTPS. JMS is a standard API for interfacing with message transport systems. Using JMS, a framework can utilize any message transport mechanism that provides a JMS interface. The J2EE Connector architecture standard does not specify a message transport mechanism, but an application integration framework can specify such a transport mechanism.

An integration framework can be based on an existing standard infrastructure, such as an application server that supports J2EE, JMS, and the J2EE Connector architecture. Using such a standard infrastructure also provides for high availability and scalability, such as by clustering and resource pooling. The framework can provide for universal connectivity by enabling the construction of XML-based application adapters that can connect to any legacy and packaged application. An adapter development kit can be used to allow users such as customers, system integrators, and packaged application vendors to quickly develop J2EE connector architecture-compliant and integration framework-based adapters. The framework can utilize XML, which means that the same data format can be used for both within- and between-enterprise integration, since many e-commerce systems use XML as the standard message format.

An integration framework can also utilize a business-process engine to allow non-programmers to graphically construct and maintain business processes. An integration framework can implement a common model on top of the J2EE Connector architecture that is focused on business-level concepts. This model, which can consist of XML-encoded events and services, allows the management of a consistent integration environment, regardless of the interface required between adapters and their target applications. The business processes can react to events generated by applications, and they can invoke an application's functionality via services that are exposed by an application adapter.

Adapter Development

XML development tools, such as may be included in an ADK, can be considered part of a metadata support layer for a design-time framework. These tools, which can comprise an XML Toolkit, can include an XML schema API, which can be characterized by a Schema Object Model (SOM). This API can be used to programmatically build XML schemas. An adapter can call an enterprise system or EIS for specific request and response metadata, which can then be programmatically transformed into an XML schema. The SOM is a set of tools that can extract many of the common details, such as syntactical complexities of XML schema operations so that a user can focus on its more fundamental aspects. Another tool that can be included is an XML Document API, which can be characterized by IDocument. This API can provide an x-path interface to a document object model (DOM) document.

An IDocument, which can facilitate XML input and output from the CCI layer in an adapter, is a higher-order wrapper around the W3C Document Object Model (DOM). The primary value-add of the IDocument

interface is that it provides an XPath interface to elements in an XML document. In other words, IDocument objects are queryable and updatable using XPath strings. For example, the following XML document describes a person named "Bob" and some of the details about "Bob."

```
<Person name="Bob">
    <Home squareFeet="2000"/>
    <Family>
       <Child name="Jimmy">
             <Stats gender="male" hair="brown" eyes="blue"/>
       </Child>
       <Child name="Susie">
             <Stats gender="female" hair="blonde" eyes="brown"/>
       </Child>
    </Family>
</Person>
```

By using IDocument, Jimmy's hair color can be retrieved using code such as:

```
System.out.println("Jimmy's hair color: " +
  person.getStringFrom("//Person[@name=\"Bob\"]/Family/Child
  [@name=\"Jimmy\"]/Stats/@hair");
```

On the other hand, if DOM is used it would be necessary to use code such as:

```
String strJimmysHairColor = null;
org.w3c.dom.Element root = doc.getDocumentElement();
if (root.getTagName().equals("Person") &&
   root.getAttribute("name").equals("Bob") {
    org.w3c.dom.NodeList list = root.
      getElementsByTagName("Family");
   if (list.getLength() > 0) {
    org.w3c.dom.Element family = (org.w3c.dom.
      Element)list.item(0);

    org.w3c.dom.NodeList childList = family.getElementsByTagName("Child");
    for (int i=0; i < childList.getLength(); i++) {
   org.w3c.dom.Element child = childList.item(i);
     if (child.getAttribute("name").equals("Jimmy")) {
   org.w3c.dom.NodeList statsList =
     child.getElementsByTagName("Stats");
     if (statsList.getLength() > 0) {
   org.w3c.dom.Element stats = statsList.item(0);
       strJimmysHairColor = stats.getAttribute("hair");
     }
    }
```

```
    }
  }
}
```

XCCI Design Pattern

A common design pattern that emerges when using an XCCI approach is to support the definition of services in an interaction implementation. In other words, a javax.resource.cci.Interaction implementation for an adapter can allow a client program to retrieve metadata from an underlying EIS in order to define an Integration service. Specifically, this means that the interaction must be able to generate the request and response XML schemas and additional metadata for a service. Additionally, the Interaction could also allow a client program to browse a catalog of functions provided by the EIS. This approach facilitates a thin client architecture for an adapter.

Data Transformation Method

Data transformation is the process of taking data from an enterprise system and transforming the data into an XML schema that can be read by the application server. For each event, a schema can define what the XML output looks like. This can accomplished by using SOM and IDocument class libraries.

The following sample code listings show one data transformation sequence. The following code can be used to transform data from an EIS into an XML schema:

```
SOMSchema schema = new SOMSchema();
SOMElement root = new SOMElement("SENDINPUT");
SOMComplexType mailType = new SOMComplexType();
root.setType(mailType);
SOMSequence sequence = mailType.addSequence();
```

```
SOMElement to = new SOMElement("TO");
to.setMinOccurs("1");
to.setMaxOccurs("unbounded");
sequence.add(to);
SOMElement from = new SOMElement("FROM");
from.setMinOccurs("1");
from.setMaxOccurs("1");
sequence.add(from);
SOMElement cc = new SOMElement("CC");
cc.setMinOccurs("1");
cc.setMaxOccurs("unbounded");
sequence.add(cc);
SOMElement bcc = new SOMElement("BCC");
bcc.setMinOccurs("1");
bcc.setMaxOccurs("unbounded");
sequence.add(bcc);
SOMElement subject = new SOMElement("SUBJECT");
subject.setMinOccurs("1");
subject.setMaxOccurs("1");
sequence.add(bcc);
SOMElement body = new SOMElement("BODY");
if (template == null)
   { body.setMinOccurs("1");
     body.setMaxOccurs("1");
   }else
     { Iterator iter = template.getTags();
       if (iter.hasNext())
       { SOMComplexType bodyComplex = new SOMComplexType();
         body.setType(bodyComplex);
        SOMAll all = new SOMAll();
       while (iter.hasNext())
       { SOMElement eNew = new SOMElement((String)iter.next());
         all.add(eNew);
       }//endwhile
        bodyComplex.setGroup(all);
       }//endif
    }//endif
sequence.add(body);
schema.addElement(root);
```

The following example shows an XML schema created by the above code:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="SENDINPUT">
```

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="TO" maxOccurs="unbounded"
      type="xsd: string"/>
    <xsd:element name="FROM" type="xsd:string"/>
    <xsd:element name="CC" maxOccurs="unbounded"
      type="xsd:string"/>
    <xsd:element name="BCC" maxOccurs=
      "unbounded" type= "xsd:string"/>
    <xsd:element name="BCC" maxOccurs="unbounded"
      type="xsd:string"/>
    <xsd:element name="BODY" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

The following example shows a valid XML document created by the schema shown above:

```
</xsd:schema>
<?xml version="1.0"?>
<!DOCTYPE SENDINPUT>
<SENDINPUT>
 <TO/>
 <FROM/>
 <CC/>
 <BCC/>
 <BCC/>
 <BODY/>
</SENDINPUT>  <xsd:schema  xmlns:xsd=
"http://www.w3.org/2001/XMLSchema">
```

The adapter can be tested and then deployed. An adapter can be deployed manually, or through a tool such as a server or integration console.

XML Toolkit

An XML toolkit can be utilized that can help to develop valid XML documents to transmit information from a resource or enterprise system to an application on the other side of an adapter. The toolkit can incorporate many of the operations required for XML manipulation into a single location, relieving the user of these often tedious chores.

An XML toolkit can be comprised primarily of two Java packages, such as a com.document package and a com.schema package. These packages can include complete Javadocs for each class, interface, and method.

IDOC

An IDocument, or IDoc, is a container that combines the W3C Document Object Model (DOM) with an XPath interface to elements in an XML document. This combination can make IDocument objects queryable and updatable simply by using XPath strings. These strings can eliminate the need to parse through an entire XML document to find specific information by allowing a user to specify just the elements the user wants to query and then returning the values of those queries.

An IDocument can be a public interface that represents an XML document. An IDocument can provide a document type name, which can

be used to refer to a description of its structure and usage. In addition, IDocuments can be mutable, and can provide methods for retrieving, modifying, adding, and removing data elements. All IDocument objects can be queryable, and can be updatable such as by using XPath strings. Also, IDocument instances can be serializable to, and un-serializable from, XML. IDocument objects can implement a Java interface such as java.io.Serializable.

The default representation of the document represented by an IDocument can be a W3C DOM instance, such as org.w3c.dom.Document. The IDocument may at times be in an unparsed state. This can happen if the IDocument was explicitly constructed this way, by using a method such as DocumentFactory.createDocument(String, boolean), for example, where the boolean argument is 'true.' The internal state of an IDocument can be determined using a method such as isParsed(). If this returns true, then the content of the document has been parsed into a DOM. If 'false' is returned, the IDocument may contain just the raw XML text.

In cases where an IDocument is in an unparsed state, the content can be parsed using a SAX parsing scheme that allows a user to build an in-memory representation of the document that is not DOM. A method such as fromXML(ContentHandler) can be used for this purpose, allowing a user to parse the document's content manually using an interface such as an implementation of a ContentHandler interface. A user could achieve similar results by getting the raw content from the IDocument and creating/invoking a custom SAX parser.

IDocument can simplify the code necessary to query and find information in a document, as opposed to code such as DOM code. For example, the following XML document describes a person named "Bob" and some of the details about "Bob":

```
<Person name="Bob">
    <Home squareFeet="2000"/>
    <Family>
        <Child name="Jimmy">
            <Stats sex="male" hair="brown" eyes="blue"/>
        </Child>
        <Child name="Susie">
            <Stats sex="female" hair="blonde" eyes="brown"/>
        </Child>
    </Family>
</Person>
```

In order to retrieve Jimmy's hair color from the <child> element by using IDocument, an XPath string can be created that seeks exactly that information, as shown the following IDocument data retrieval code sample:

```
System.out.println("Jimmy's hair color: " + person.getStringFrom
("//Person[@name=\"Bob\"] /Family/Child[@name=\"Jimmy\"]/Stats/@hair");
```

Schema Object Model (SOM)

A schema object model (SOM) is an interface useful for programmatically building schemas, such as XML schemas. SOM can comprise a set of tools that can extract and validate many of the common details, such as syntactical complexities of schema, so that a user can focus on more fundamental aspects. As shown in the method of **Figure 3**, for example, an XML schema can be created for a client application using a schema object model **300**. A component such as a resource adapter can call into an EIS for specific request/response metadata. Metadata can be received from the EIS in response to the request **302**, which can be programmatically transformed into an XML document that conforms to the XML schema for the client application **304**. At least a portion of the XML

document can be validated against the XML schema for the client application **306**. An IDocument interface can be used to parse and/or retrieve elements or portions from the XML document in order to validate those elements or portions. Once the XML document is validated, it can be passed to the client application as a response document **308**.

A flowchart for such a method is shown in **Figure 4**. The request is received from the client application to an EIS **400**. The EIS returns metadata in response to the request **402**. The metadata is transformed into an XML document **404**. At least portions of the XML document can be parsed or extracted, such as by using an IDocument interface, and can be compared to the XML schema for the client application in order to validate those portions **406**. A determination is made as to whether each portion is valid **408**. If each portion is valid, the validated XML document can be returned to the client as a response document **410**. If each portion is not valid, a determination should be made whether the client application should receive invalid documents **412**. If the client should receive an invalid document, each error can be logged **414** and the validated (or invalidated) XML document is passed to the client as a response document **410**.

If the client should not receive an invalid XML document, a determination should be made whether an end condition has been met **416**. An end condition can be set so that an infinite loop is not created if the metadata cannot be transformed into a valid XML document. "Valid" in this instance means that the XML document is valid against the XML schema, not that the document is a valid XML document. The client application may receive a document that conforms to the XML schema but that is not a proper XML document. If the end condition has not been met,

another attempt can be made to transform the metadata into an XML document **404**. If the end condition has been met, such as a number of iterations or timeout period end being reached, an error can be returned to the client application and processing of this request can stop **418**.

An XML schema is like a contract between the EIS and an application on the other side of the adapter. This contract can specify how data coming from the EIS must appear in order for the application to manipulate it. A document, or an XML-rendered collection of metadata from an EIS, can be considered valid if it meets the rules specified in the schema, regardless of whether or not the XML is correct. For example, if a schema required a name to appear in a <name> element and that element required two child elements, such as <firstname> and <lastname>, to be valid, the document from the EIS would have to appear in a form such as:

```
<name>
   <firstname>Joe</firstname>
   <lastname>Smith</lastname>
</name>
```

and the schema would have to appear in a form such as:

```
<schema>
  <element name="name">
    <complexType>
      <sequence>
        <element name="firstname" />
        <element name="lastname" />
      </sequence>
    </complexType>
  </element>
</schema>
```

No other form of <name></name>, such as "<name>Joe Smith</name>" would be valid, even though the XML is correct.

Creating the Schema

An XML schema can be created programmatically by using the classes and methods provided with SOM. One benefit of using such a tool is that it allows a user to tailor a schema for that user's needs simply by populating the variables in the program components. For instance, the following code examples create a schema that validates a purchase order document:

```
import com.bea.schema.*;
import com.bea.schema.type.SOMType;

  public class PurchaseOrder
  {
    public static void main(String[] args)
    {
      System.out.println(getSchema().toString());
    }


    public static SOMSchema getSchema()
    {
      SOMSchema po_schema = new SOMSchema();

    po_schema.addDocumentation("Purchase order schema for
     Example.com.\nCopyright 2000 Example.com.\nAll rights
     reserved.");

      SOMElement purchaseOrder =
        po_schema.addElement("purchaseOrder");

      SOMElement comment = po_schema.addElement("comment");

SOMComplexType usAddress =
  po_schema.addComplexType("USAddress");

SOMSequence seq2 = usAddress.addSequence();
```

```
// adding an  object to a SOMSchema defaults to type="string"
seq2.addElement("name");
seq2.addElement("street");
seq2.addElement("city");
seq2.addElement("state");
seq2.addElement("zip", SOMType.DECIMAL);
```

One benefit to such a tool is that it is only necessary to populate the variables in the program components to tailor a schema for particular needs. Attributes can be set in the same way that elements are created, such as:

```
SOMAttribute country_attr =
        usAddress.addAttribute("country",
        SOMType.NMTOKEN);
        country_attr.setUse("fixed");
        country_attr.setValue("US");
```

To correctly set these attributes, their addressibility should be maintained. Like complexTypes, simpleTypes can be added to the root of the schema, such as:

```
SOMSimpleType skuType = po_schema.addSimpleType("SKU");
SOMRestriction skuRestrict = skuType.addRestriction
  (SOMType.STRING);
skuRestrict.setPattern("\\d{3}-[A-Z]{2}");
   SOMComplexType poType =
     po_schema.addComplexType("PurchaseOrderType");
   purchaseOrder.setType(poType);
   poType.addAttribute("orderDate", SOMType.DATE);
```

The addSequence() method of a SOMComplexType object can return a SOMSequence reference, allowing a user to modify the element that was added to the element. In this way objects can be added to the schema, such as by implementing an addSequence() method to modify an element:

```
SOMSequence poType_seq = poType.addSequence();
```

```
poType_seq.addElement("shipTo", usAddress);
poType_seq.addElement("billTo", usAddress);
```

Attributes of an element within a schema can be set by calling setter methods of a SOMElement object. For example,.an the implementation of setMinOccurs() and setMaxOccurs() can be given by:

```
SOMElement commentRef = new SOMElement(comment);
  commentRef.setMinOccurs(0);
  poType_seq.add(commentRef);
SOMElement poType_items = poType_seq.addElement("items");
  SOMComplexType itemType = po_schema.addComplexType("Items");
  SOMSequence seq3 = itemType.addSequence();
  SOMElement item = new SOMElement("item");
    item.setMinOccurs(0);
    item.setMaxOccurs(-1);
    seq3.add(item);
  SOMComplexType t = new SOMComplexType();
    item.setType(t);
  SOMSequence seq4 = t.addSequence();
    seq4.addElement("productName");
  SOMElement quantity = seq4.addElement("quantity");
  SOMSimpleType st = new SOMSimpleType();
    quantity.setType(st);
  SOMRestriction restrict =
    st.addRestriction(SOMType.POSITIVEINTEGER);
    restrict.setMaxExclusive("100");
```

In this example, the items element for PurchaseOrderType was created before Items type. The reference can be created and the type set once the Items type object is available, such as by using:

```
poType_items.setType(itemType);
```

An element can also be added to the schema. Adding an element can be done by implementing an addElement() method of SOMSequence, or the add() method from a previously created SOMElement. These methods can be shown by:

```
seq4.addElement("USPrice", SOMType.DECIMAL);
  SOMElement commentRef2 = new SOMElement(comment);
    commentRef2.setMinOccurs(0);
    seq4.add(commentRef2);
  SOMElement shipDate = new SOMElement("shipDate", SOMType.DATE);
    shipDate.setMinOccurs(0);
    seq4.add(shipDate);
  t.addAttribute("partNum", skuType);
    return po_schema;
  }
}
```

When running the code shown in the previous examples, the following schema can be created:

```
<?xml version="1.0" ?>
<!DOCTYPE schema (View Source for full doctype...)>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/XMLSchema">
  <xsd:annotation>
    <xsd:documentation>Purchase order schema for Example.com.
      Copyright 2000 Example.com. All rights
      reserved.</xsd:documentation>
  </xsd:annotation>
  <xsd:simpleType name="SKU">
    <xsd:annotation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      <xsd:element type="USAddress" name="shipTo" />
      <xsd:element type="USAddress" name="billTo" />
```

```
      <xsd:element ref="comment" minOccurs="0" />
      <xsd:element type="Items" name="items" />
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date" />
</xsd:complexType>

<xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" name="item"
      minOccurs="0">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element type="xsd:string"
            name="productName"/>
          <xsd:element name="quantity">
            <xsd:simpleType>
              <xsd:restriction base=
                "xsd:positiveInteger">
                <xsd:maxExclusive value="100"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element type="xsd:decimal" name=
            "USPrice" />
          <xsd:element ref="comment"
            minOccurs="0" />
          <xsd:element type="xsd:date"
            name="shipDate" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="partNum" type="SKU" />
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="USAddress">
 <xsd:sequence>
    <xsd:element type="xsd:string" name="name" />
    <xsd:element type="xsd:string" name="street" />
    <xsd:element type="xsd:string" name="city" />
    <xsd:element type="xsd:string" name="state" />
    <xsd:element type="xsd:number" name="zip" />
 </xsd:sequence>
   <xsd:attribute name="country" use="fixed" value="US"
     type="xsd:NMTOKEN" />
</xsd:complexType>
<xsd:element type="PurchaseOrderType" name="purchaseOrder" />
<xsd:element type="xsd:string" name="comment" />
</xsd:schema>
```

Validating an XML Document

Once the schema is created, it can be used to validate a document sent from an EIS. SOM can be used to validate XML DOM documents by using a SOMSchema method such as isValid(). SOMElement can have a corresponding isValid() method for validating an element instead of the DOM document. The isValid() method can determine if 'document' or 'element' is valid, and if not, can compile a list of the errors. If the document is valid, isValid() can return 'true' and the list of errors can be empty.

An isValid() method can be implemented in a number of different ways, including the following ways:

```
public boolean isValid(org.w3c.dom.Document doc,
                       java.util.List errorList)
public boolean isValid(IDocument doc,
                       List errorList)
```

In this example, "doc" refers to the document instance to be validated, and "errorList" refers to a list of errors found in the document and/or doc.

The isValid() method can return a boolean value of 'true' if the document is valid with respect to this schema. If the document is not valid with respect to the schema, isValid() can return 'false' and the errorList can be populated. The errorList is a java.util.List for reporting errors found in the document, doc. The error list is cleared before validating the document. Therefore, the list implementation used must support the clear() method. If isValid() returns false, the error list is populated with a list of errors found during the validation procedure. The items in the list are instances of the class com.bea.schema.SOMValidationException. If isValid() returns true,

errorList is empty. The following shows an example of an isValid() implementation:

```
SOMSchema schema = ...;
IDocument doc = DocumentFactory.createDocument
        (new FileReader(f));
java.util.LinkedList errorList = new
        java.util.LinkedList();
boolean valid = schema.isValid(doc, errorList);...
if (! valid){
        System.out.println("Document was invalid.
            Errors were:");

    for (Iterator i = errorList.iterator; i.hasNext();)
{
    System.out.println(((SOMValidationException)
        , i.next).toString());
}
```

The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to one of ordinary skill in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.

What is claimed is:

1. A method for passing communication between an enterprise system and a client application, comprising:

transforming metadata received from an enterprise system into an XML document;

validating at least a portion of the XML document against an XML schema for the client application; and

passing the XML document to the client application.

2. A method according to claim 1, further comprising:

building the XML schema using an XML schema API.

3. A method according to claim 1, further comprising:

building the XML schema using a schema object model.

4. A method according to claim 3, wherein:

building the XML schema further includes using classes and methods in the schema object model.

5. A method according to claim 2, wherein:

building the XML schema includes populating variables in program components of the schema object model in order to tailor the XML schema for the client application.

6. A method according to claim 1, further comprising:

creating a schema object model to validate an XML schema for the client application.

7. A method according to claim 1, wherein:

compiling a list of errors containing elements of the XML document that are not valid.

8. A method according to claim 1, further comprising:

translating information passing between the enterprise system and the client application using a resource adapter.

9. A system for passing communication between an enterprise system and a client application, comprising:

a resource adapter adapted to receive metadata from an enterprise system; and

an XML schema component adapted to transform the metadata into an XML document and validate the XML document against an XML schema;

wherein the resource adapter is further adapted to pass a validated XML document to the client application.

10. A system according to claim 9, further comprising:

a schema object model adapted to provide the ability to create the XML schema.

11. A system according to claim 9, further comprising:

an application view component adapted to provide an interface to the enterprise system for the client application.

12. A computer-readable medium, comprising:

means for transforming metadata received from an enterprise system into an XML document;

means for validating at least a portion of the XML document against an XML schema for the client application; and

means for passing the XML document to the client application.

13. A computer program product for execution by a server computer for formatting enterprise data for an application, comprising:

computer code for transforming metadata received from an enterprise system into an XML document;

computer code for validating at least a portion of the XML document against an XML schema for the client application; and

computer code for passing the XML document to the client application.

14. A system for formatting enterprise data for an application, comprising:

means for transforming metadata received from an enterprise system into an XML document;

means for validating at least a portion of the XML document against an XML schema for the client application; and

means for passing the XML document to the client application.

15. A computer system comprising:

a processor;

object code executed by said processor, said object code configured to:

transform metadata received from an enterprise system into an XML document;

validate at least a portion of the XML document against an XML schema for the client application; and

pass the XML document to the client application.

16. A computer data signal embodied in a transmission medium, comprising:

a code segment including instructions to transform metadata received from an enterprise system into an XML document;

a code segment including instructions to validate at least a portion of the XML document against an XML schema for the client application; and

a code segment including instructions to pass the XML document to the client application.

17. A method for passing communication between an enterprise system and a client application, comprising:

transforming metadata received from an enterprise system into an XML document;

querying the XML document using a document interface component in order to extract at least a portion of the XML document;

validating said at least a portion of the XML document against an XML schema for the client application; and

passing the XML document to the client application.

18. A method according to claim 17, further comprising:

building the XML schema using an XML schema API.

19. A method according to claim 17, further comprising:

building the XML schema using a schema object model.

20. A method according to claim 19, wherein:

building the XML schema further includes using classes and methods in the schema object model.

21. A method according to claim 18, wherein:

building the XML schema includes populating variables in program components of the schema object model in order to tailor the XML schema for the client application.

22. A method according to claim 17, further comprising:

creating a schema object model to validate an XML schema for the client application.

23. A method according to claim 17, wherein:

compiling a list of errors containing elements of the XML document that are not valid.

24. A method according to claim 17, further comprising:

translating information passing between the enterprise system and the client application using a resource adapter.

25. A method for passing communication between an enterprise system and a client application, comprising:

transforming metadata received from an enterprise system into an XML document;

querying a document interface component in order to validate at least a portion of the XML document against an XML schema; and

passing the XML document to the client application.

26. A system for passing communication between an enterprise system and a client application, comprising:

a resource adapter adapted to receive metadata from an enterprise system in response to a request from a client application;

a document interface component adapted to allow the querying of elements in an XML document; and

an XML schema component adapted to transform the metadata into an XML document and validate elements the XML document against an XML schema, the XML schema component obtaining the elements through the document interface component;

wherein the resource adapter is further adapted to pass a validated XML document to the client application.

27. A system according to claim 26, wherein:

the document interface component is an XML document API.

28. A system according to claim 26, wherein:

the document interface component is adapted to work with DOM documents.

29. A system according to claim 26, wherein:

the document interface component provides an XPath interface to elements in an XML document.

30. A system according to claim 26, wherein:

the document interface component allows portions of an XML document to be queried and updated using XPath strings.

31. A system according to claim 26, wherein:

the document interface component comprises a container including a DOM instance and an XPath interface.

32. A system according to claim 26, wherein:

the document interface component is a public interface that represents an XML document.

33. A system according to claim 26, wherein:

the document interface component allows manipulation of elements in an XML document, the manipulation selected from the group consisting of retrieving, modifying, adding, and removing data elements.

34. A system according to claim 26, further comprising:

a schema object model adapted to provide the ability to create the XML schema.

35. A system according to claim 26, further comprising:

an application view component adapted to provide an interface to the enterprise system for the client application.

36. A computer-readable medium, comprising:

means for transforming metadata received from an enterprise system into an XML document;

means for querying the XML document using a document interface component in order to extract at least a portion of the XML document;

means for validating said at least a portion of the XML document against an XML schema for the client application; and

means for passing the XML document to the client application.

37. A computer program product for execution by a server computer for formatting enterprise data for an application, comprising:

computer code for transforming metadata received from an enterprise system into an XML document;

computer code for querying the XML document using a document interface component in order to extract at least a portion of the XML document;

computer code for validating said at least a portion of the XML document against an XML schema for the client application; and

computer code for passing the XML document to the client application.

38. A system for formatting enterprise data for an application, comprising:

means for transforming metadata received from an enterprise system into an XML document;

means for querying the XML document using a document interface component in order to extract at least a portion of the XML document;

means for validating said at least a portion of the XML document against an XML schema for the client application; and

means for passing the XML document to the client application.

39. A computer system comprising:

a processor; object code executed by said processor, said object code configured to:

transform metadata received from an enterprise system into an XML document;

query the XML document using a document interface component in order to extract at least a portion of the XML document;

validate said at least a portion of the XML document against an XML schema for the client application; and

pass the XML document to the client application.

40. A computer data signal embodied in a transmission medium, comprising:

a code segment including instructions to transform metadata received from an enterprise system into an XML document;

a code segment including instructions to query the XML document using a document interface component in order to extract at least a portion of the XML document;

a code segment including instructions to validate said at least a portion of the XML document against an XML schema for the client application; and

a code segment including instructions to pass the XML document to the client application.

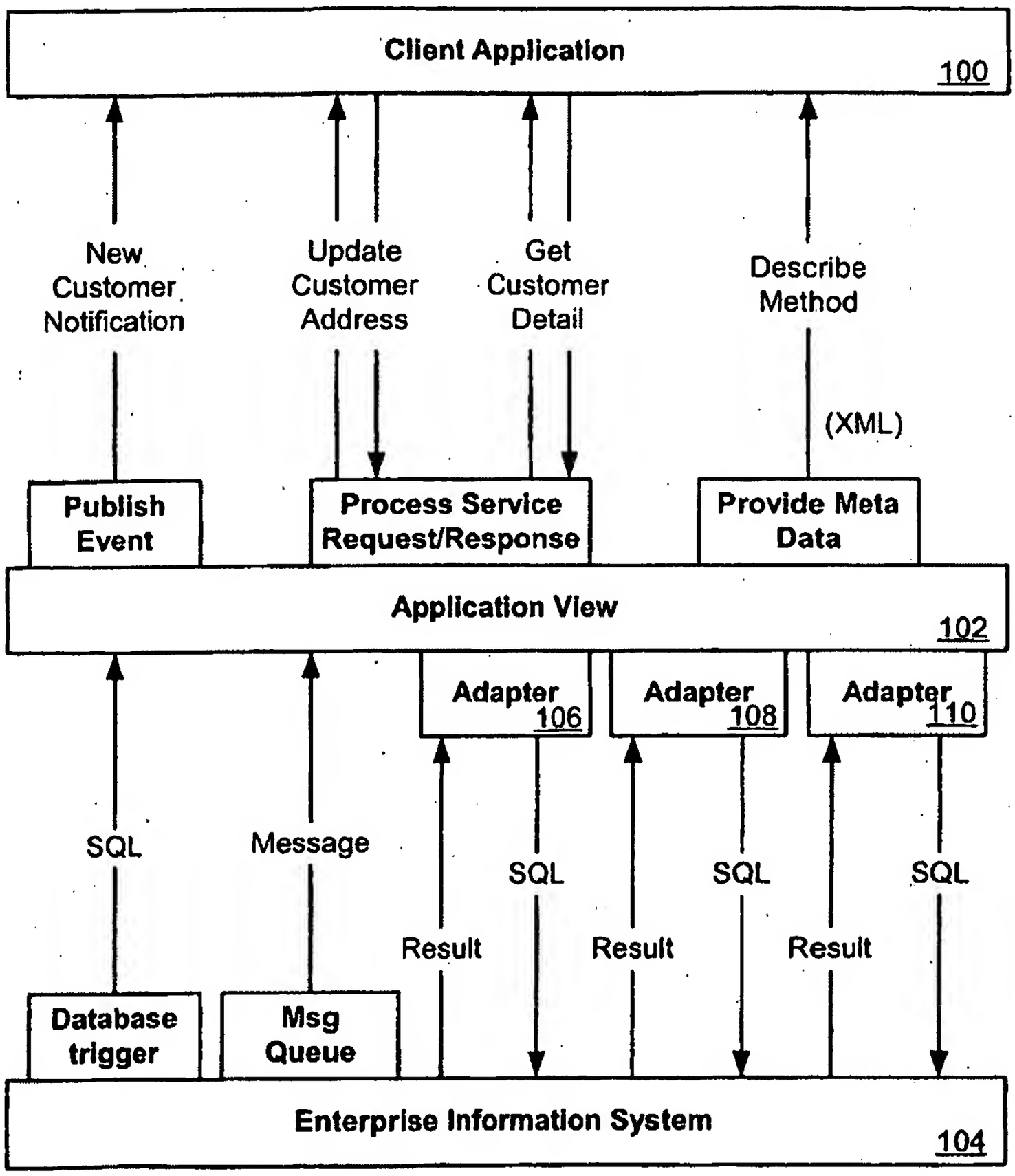

Client Application
100
New Customer Notification
Update Customer Address
Get Customer Detail
Describe Method
(XML)
Publish Event
Process Service Request/Response
Provide Meta Data
Application View
102
Adapter 106
Adapter 108
Adapter 110
SQL
Message
SQL
SQL
SQL
Result
Result
Result
Database trigger
Msg Queue
Enterprise Information System
104

Figure 1

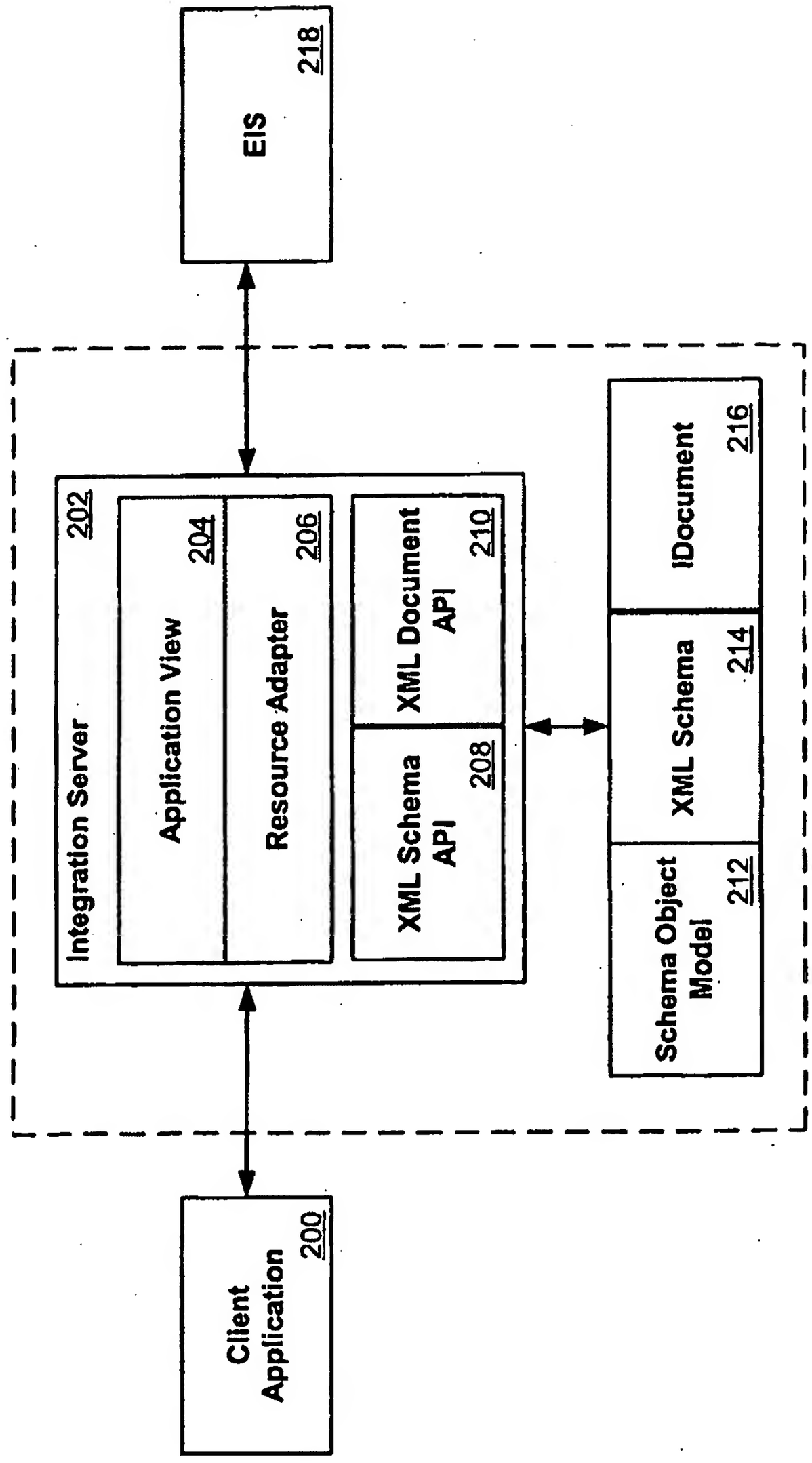

EIS
218
Integration Server
202
Application View
204
Resource Adapter
206
XML Schema API
208
XML Document API
210
Schema Object Model
212
XML Schema
214
IDocument
216
Client Application
200

Figure 2

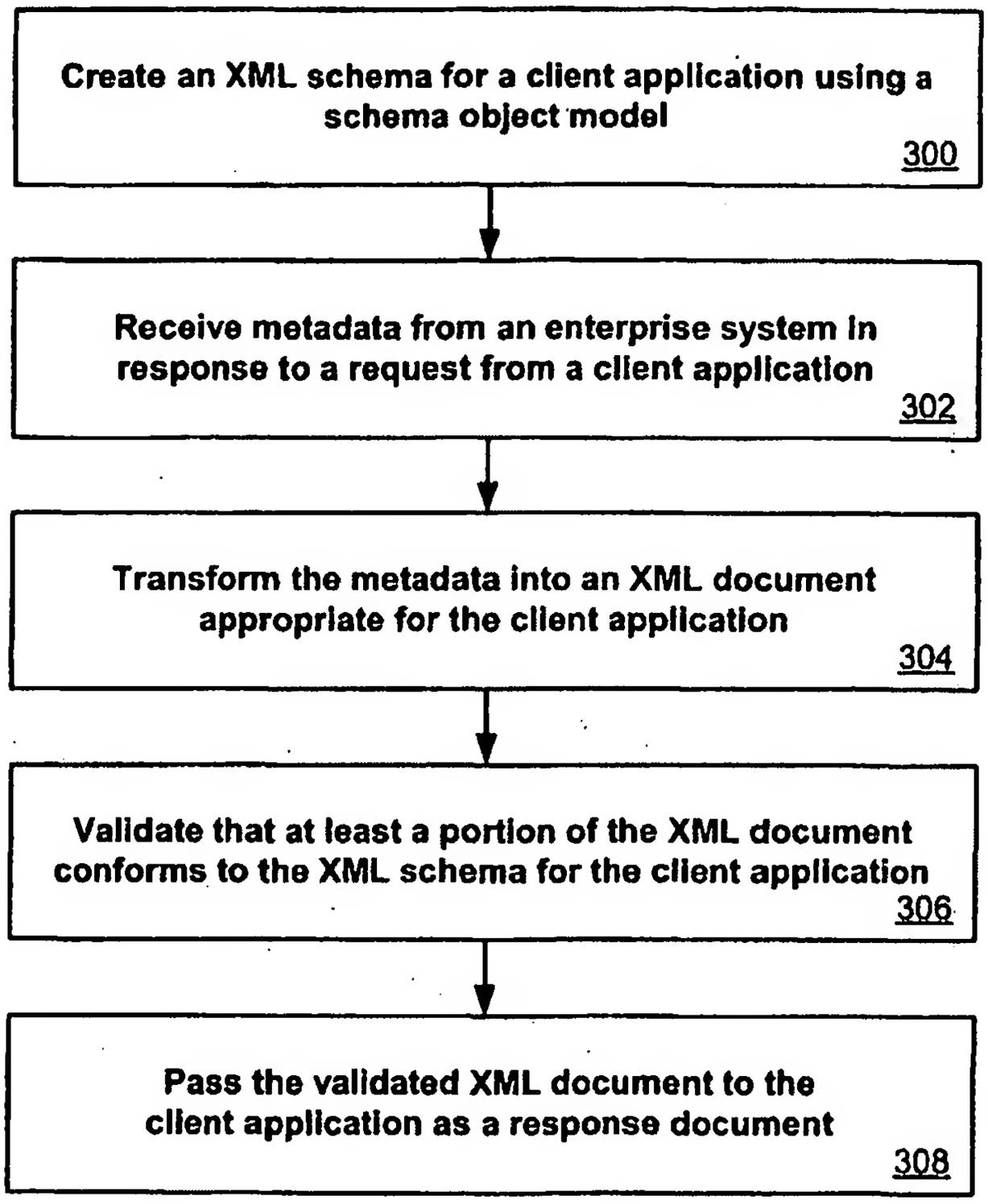

Create an XML schema for a client application using a schema object model
300
Receive metadata from an enterprise system in response to a request from a client application
302
Transform the metadata into an XML document appropriate for the client application
304
Validate that at least a portion of the XML document conforms to the XML schema for the client application
306
Pass the validated XML document to the client application as a response document
308

Figure 3

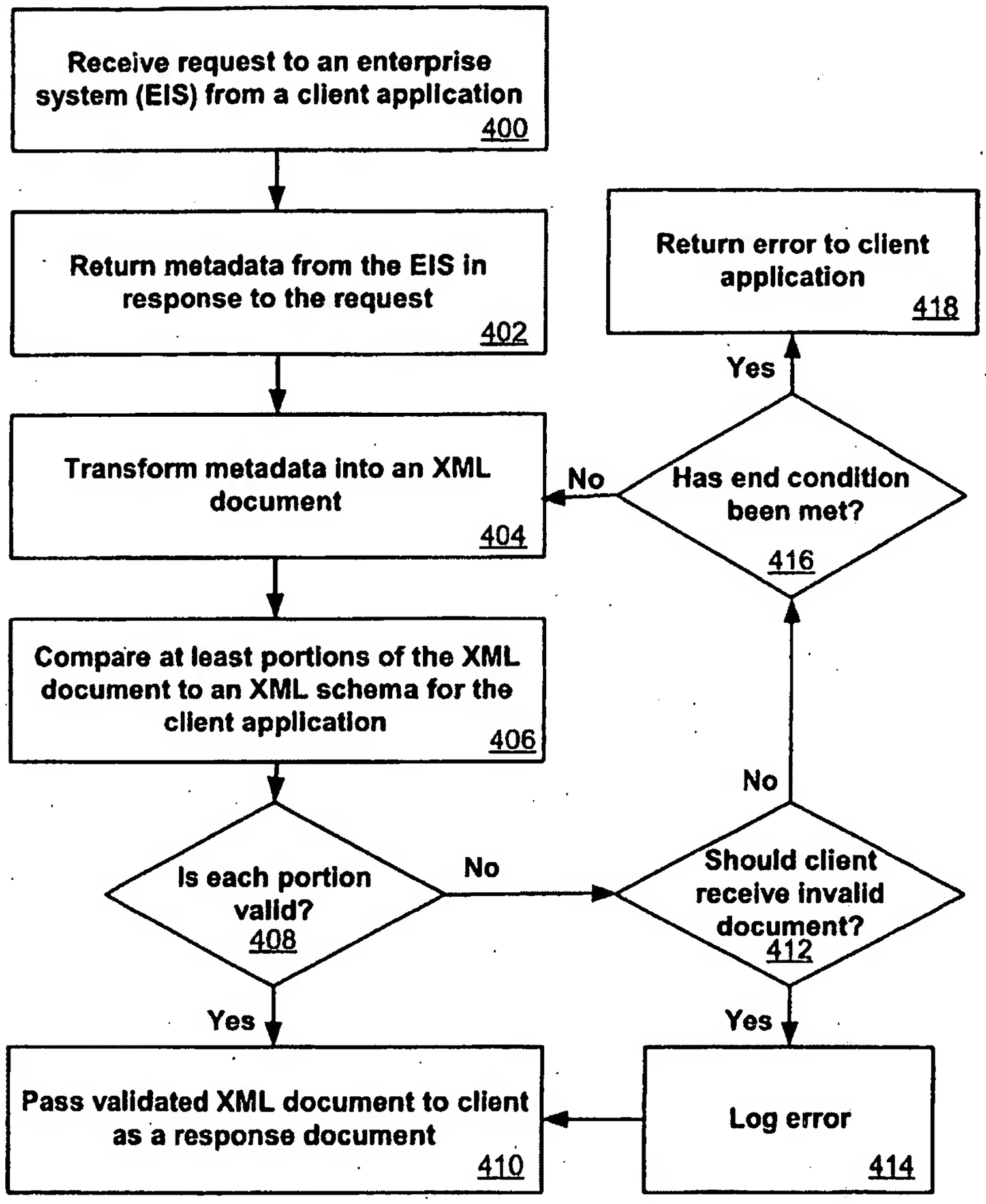

Receive request to an enterprise system (EIS) from a client application
400
Return metadata from the EIS in response to the request
402
Transform metadata into an XML document
404
Compare at least portions of the XML document to an XML schema for the client application
406
Is each portion valid?
408
No
Yes
Pass validated XML document to client as a response document
410
Should client receive invalid document?
412
No
Yes
Log error
414
Has end condition been met?
416
No
Yes
Return error to client application
418

Figure 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/33090

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 12/00, 17/30
US CL : 707/513
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 707/513

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
IEEE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Continuation Sheet

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 6,226,675 B1 (MELTZER et al) 01 May 2001 (01.05.2001), column 3, lines 28-41, column 7, lines 55-61, column 22, lines 39-51, column 23, lines 43-46, column 25, lines 28-51, column 26, lines 1-9, column 30, lines 43-45, column 79, lines 47-52.	1-40
Y	US 6,154,738 A (CALL) 28 November 2000 (28.11.2000), column 24, lines 45-59, column 25, lines 26-34, column 32, lines 20-25.	1-40

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 January 2003 (13.01.2003)	11 FEB 2003
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer Heather Herndon Telephone No. (703) 305-4700

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

PCT/US02/33090

Continuation of B. FIELDS SEARCHED Item 3:
EAST
search terms: XML, DOM, object model, transformation, translation, conversion, converting, schema, validation, metadata